NASA TECHNICAL NOTE

NASA TN D-2863

NASA TN D-2863

NASA PERT TIME II

by Ross C. Bainbridge and Elizabeth Ryan

Lewis Research Center

Cleveland, Ohio

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • JUNE 1965

NASA TN D-2863

NASA PERT TIME II

By Ross C. Bainbridge and Elizabeth Ryan

Lewis Research Center
Cleveland, Ohio

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

For sale by the Clearinghouse for Federal Scientific and Technical Information
Springfield, Virginia 22151 – Price $1.00

NASA PERT TIME II

by Ross C. Bainbridge and Elizabeth Ryan

Lewis Research Center

SUMMARY

25598

The NASA PERT Time II program has been produced to answer the need for a compiler language PERT time program. Flexibility, adaptability, and efficiency are the prime considerations of the NASA PERT Time II program. This program is designed to be compatible with current and future data processing equipment configurations in use throughout both government and industry. It is also compatible with various standards of input/output and is designed to cross computer manufacturing lines. The program can easily be modified to change the approach of analyzing PERT time networks and the methods of reporting the results of these analyses. The capacity of the program is in excess of 30 000 activities when utilizing a modular technique.

Author

INTRODUCTION

PERT (Program Evaluation and Review Technique) time is defined as a disciplined management technique involving computer processing. (A glossary of terms is given in appendix A.) PERT time aids in the planning and control of the variables, which are time and performance, of project development. PERT gives the project manager insight into his current and future program development as well as potential problem areas related to its development. The total usefulness of PERT, however, is entirely dependent on judgement and analysis. PERT can formalize planning and serve as a guide to control, but it cannot replace good management.

The purpose of this report is to present information relating to the NASA PERT time program in Fortran IV compiler language. It includes a brief history of program development, general algorithms used or developed to make the program functional, and program systems information. The latter is designed to give computer oriented professional and technical personnel enough insight into the program functioning to allow for maintenance as well as further development of the program.

In March of 1963, the version of PERT in use by NASA and its contractors was a machine language adaptation of the Lockheed PERT time program. A combination of hardware and system developments in the computer industry and the disadvantages of maintaining a machine-coded program prompted NASA Lewis to propose that a PERT program be written entirely in compiler language. Some of these disadvantages can be summarized as follows:

(1) The machine-coded program could be run only on one manufacturer's equipment.

(2) A great deal of time was being spent in maintaining machine-coded programs and in modifying them each time a system change was implemented.

(3) The adoption of new hardware (substitution of disk or drum for tapes) by an installation even without a change of manufacturer often required extensive rewriting of the machine-coded programs.

A compiler-written program would substantially minimize these problems. Since modifications to a compiler-written program can more easily be made by recompiling a source deck, language documentation would automatically be provided for all modifications.

NASA Lewis also proposed that the PERT program be written in FORTRAN IV because it is the compiler that has been implemented by most computer manufacturers, and it is the compiler language most used by industry as well as government.

The Lewis proposal was accepted by NASA Headquarters in June of 1963. The program development was divided into two phases. The first phase was a feasibility study in which a program written in Fortran IV with a limited network capacity was produced. This program, known as the Lewis Goddard (NASA) PERT Time I program, not only proved that an efficient Fortran IV coded program could be produced but also became widely distributed throughout industry and the government. (Performance data from this program can be found in appendix B.) The second phase was to produce a program utilizing the concepts of the first phase with a network capacity of at least 30 000 activities. It is this program, known as the NASA PERT Time II program, which is being presented in this report.

IMPLEMENTATION OF A MODULAR TECHNIQUE

The increased capacity of the NASA PERT Time II program is obtained by using a subnet technique. In the PERT network shown in figure 1, where the circles represent events and the connecting lines represent activities, the shaded activities make up a subnet, as do the activities enclosed in the broken lines. Note that there are three events common to both subnets. These events are called interface events. Note that interfaces are events not activities.

In practical terms, a subnet is often a logical entity of some kind. For instance, in a network (fig. 2) representing a project involving four contractors (A, B, C, and D) there could be four subnets each representing the work assigned to one of the four contractors. It is not required that the interface events have the same event number interior to each subnet in which they appear. This eliminates the necessity of coordinating numbering of common events among many contractors. To eliminate this, each interface event is given an alphanumeric name, the interface name, when its subnet is to be integrated. In fig-

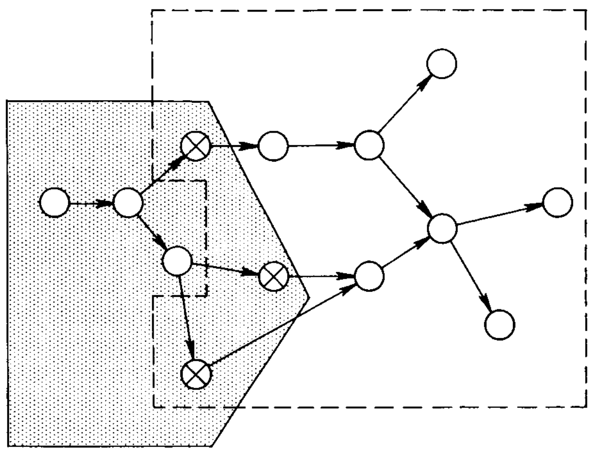
Figure 1. - PERT network.

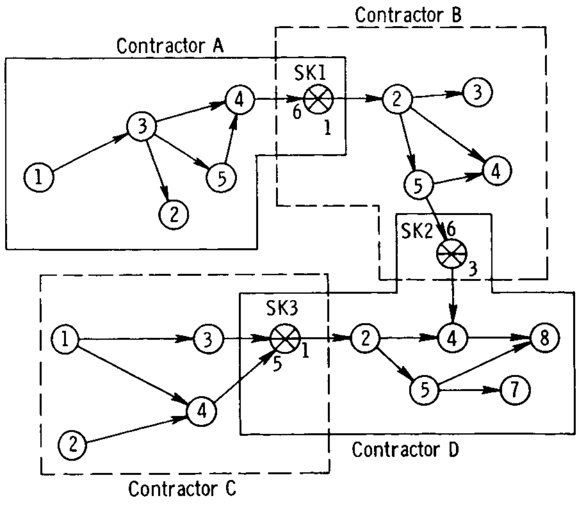

Figure 2. - Project network.

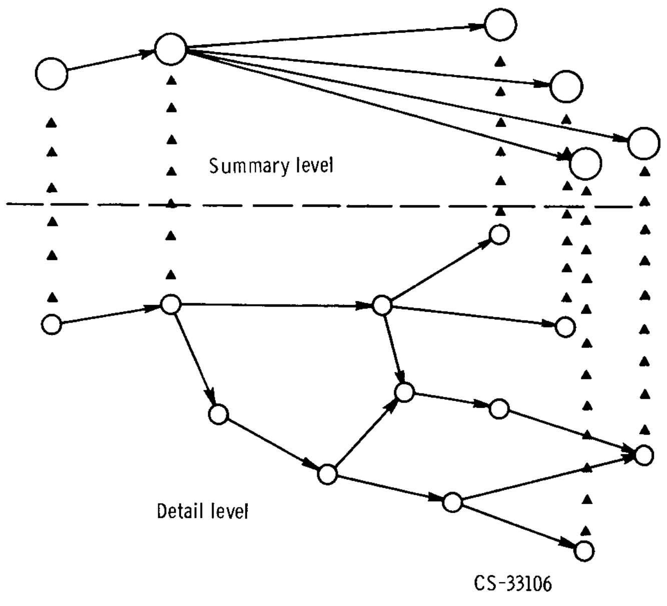

Figure 3. - Summary subnet.

ure 2, for example, the interface event that has been labeled SK1 may be known in contractor A's subnet as event 6 and as event 1 in contractor B's subnet. With reference to the network as a whole, however, it is simply interface event SK1.

PROGRAM FEATURES

A useful feature of the program is the provision for a different type of subnet, the summary network. Suppose the subnet shown in figure 3 below the dashed line is being maintained by a department for its own use. It may be that only those events with upward pointing broken arrows need to be reported to higher management. These events can then be made interfaces to a new subnet, called the summary network, whose activities are definable by the PERT analysts. The resulting summary subnet is represented by the network above the dashed line. It is a summary of the original subnet or subnets and shows not only event relation but also PERT network logical flow as indicated by the solid arrows. The program will compute time estimates along each path of the summary network by using the detailed paths from the original. If requested, activity cards for the summary network can be punched out with delta time estimates. This deck can then be sent on to higher management to be run as this department's subnet in a larger network.

Most existing PERT programs processed networks having either event or activity oriented input, but not both. The NASA PERT Time II program is primarily activity oriented, but it also provides for event oriented input. This was done

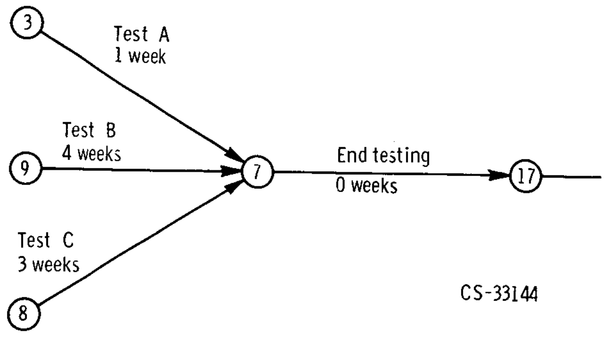

Figure 4. - Dummy activity.

to eliminate the insertion into networks of dummy activities with zero time estimates.

The activity connecting events 7 and 17 in figure 4 is an example of a dummy inserted for the express purpose of inventing a place to hang the label END TESTING. What is actually needed here is a way of identifying event 7 as the end of testing. The insertion of the dummy has added an extra event and activity to the network. This practice of substituting for event nomenclature is quite common and causes a significant increase in the size and processing time of a network.

With the PERT TIME II program, the event 7 can be named directly by the use of an event card:

```
00000070000007     END TESTING
```

The event number is entered in both the predecessor and successor columns, and nomenclature appears in the normal field. At report time, the event will appear in normal sort order with its expected and allowed dates and slack. The event card does not in any way enter into the PERT calculation.

The updating or file maintenance technique used in PERT Time II also represents a new approach. In previous PERT programs the master file has been nothing more than a tape bearing the activity cards for a given network. When it was desired to change the network, the tape was first updated to obtain a new master file, and the new master file was used as input for a complete reexecution of the network. The PERT Time II program performs updating as a part of the normal PERT run, thus eliminating duplication of operations. A binary tape developed as part of the PERT calculation is used as the master file. This tape contains not only the activity cards in packed form but also all other information needed to make reports directly from the tape without recalculation. All this information is separated by subnet. Since not all subnets may need to be changed on a given update run, only those that are changed need be recalculated. The master tape is read only once as updating and recalculation of a subnet are overlapped. In addition to providing a fast and efficient update, this technique eliminates dependence on the availability of a second computer.

As a further aid to maintaining networks, completed activities may be deleted from the master file by control card option. This feature has two important results. First, by eliminating past activities that no longer alter the project schedule, it reduces the effective in-core size of the network. Second, it has been found that a great deal of updating is done for the purpose of removing completed activities. This type of routine updating can now be completely eliminated.

PROGRAM ANALYSIS

The program can be divided functionally into four sections by considering together those subroutines concerned with network analysis, execution control, reporting, and updating of a master tape.

Network Analysis

The analysis of a subnetted network proceeds in three distinct steps or phases: the condensation of all subnets to obtain a control network, the calculation of expected and allowed dates for the control network, and the final determination of expected and allowed dates for each subnet for which a report is requested.

First phase. - At the beginning of the first phase, all interface cards and activity cards for a particular subnet have been read. The activity cards provide a table of the activities, each defined by a beginning and ending event, which make up this subnet. With each activity is associated a time estimate, a description, a scheduled or actual date if any, and other information. The interface cards provide a list of all events in this subnet that also appear in another subnet and the corresponding interface names by which each such event will be recognized whenever it appears throughout the entire network.

Subroutine TEST locates all start and end events for the subnet. If they are not interface events they are made to be, so that the entire subnet is bounded by interfaces. Subroutine TOPOL is now used to trace out all subnet paths and to make time calculations as it proceeds (see appendix C). At this point the expected and allowed dates for subnet activities cannot be determined because they are affected at interface events by other subnets. However, for all those activities on a path connecting two interface events a beginning and ending time relative to the dates which will be determined for the interface events can be calculated. Also, the greatest total time along all paths connecting any two interface events can be found and can be thought of as the estimated time for the control network activity defined by the interface events. Since a given subnet can be affected by any other only at the points of connection between the two (i.e., at the interface events), only these control network activities need be considered when relating this subnet to all others. The first phase of analysis for each subnet then results in a calculated relative time for each subnet event and a list of control network activities. The relative time calculations are placed in a separate file for each subnet. The list of control network activities from each subnet is added to a table which, after all subnets have been processed, describes the control network.

Second phase. - The control network that was constructed in the first phase of analysis is a complete network in its own right. It is an abstraction of the original network and fully reflects all interrelationships between subnets. All scheduled dates encountered in processing individual subnets have been carried through to the control network so that these, together with the time span between interface events, make it possible to calculate expected and allowed dates for each activity in the control network. The control network can, in other

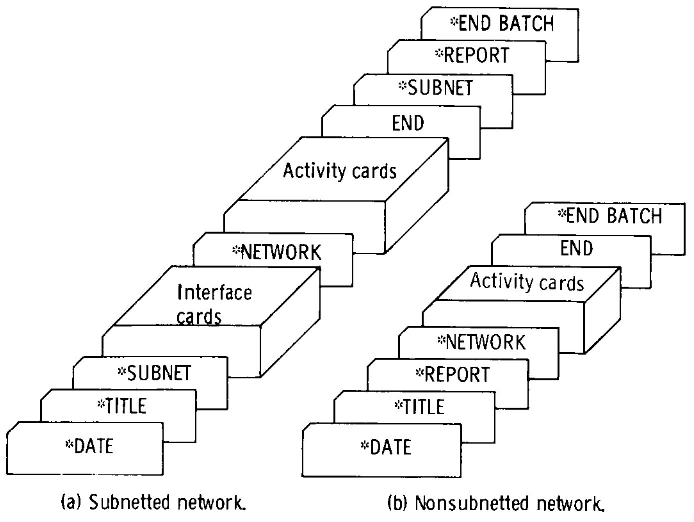

Figure 5. - Sample deck sequences.

words, be completely evaluated in the same manner as any complete nonsubnetted network. It is this evaluation that makes up the second phase of analysis. The table of activities which define the control network is first sorted by using subroutines PREPAR, SSORT, and MOVE so that all activities originating at the same interface event are together. Subroutine TEST then isolates the start and end events and calls on subroutine TOPOL to trace out all network paths and to compute expected and allowed dates for each event. The event dates are then converted to activity dates so that both event- and activity-oriented reports on the control network can be made at this time. The result of the second phase of analysis therefore is to associate with each interface event an expected and allowed date.

Third phase. - The third phase of analysis is performed only on those subnets for which a report has been requested. Since the first phase of analysis has produced activity times relative to an interface event for each subnet activity, and the second phase has determined expected and allowed dates for each interface event, the third phase need only combine these two results to give a complete description of an individual subnet. When a report of a given subnet is requested, the control section of the program locates the file which contains the relative time information for that subnet. The starts and ends for the subnet and paths have been previously determined so TOPOL can be called to trace the subnet paths and compute expected and allowed dates directly. As the dates for each activity are determined, these activity dates and all other descriptive information (slack, time remaining, etc.) are placed in a file in a packed form to conserve space. If output in predecessor order has been requested for this network, it is printed at this time. The file with packed activity information will be used by the report subroutines to perform any other outputs for this subnet.

Execution Control

The analysis and reporting of a network is controlled by the main supervisory routine ASKER in accordance with the asterisk control cards supplied by the PERT analyst. The execution of a subnetted network whose deck format is as shown in figure 5(a) will now be described.

The asterisk cards are read and interpreted one by one. The information from the *DATE and *TITLE cards that defines the date and title of the network is stored in the appropriate storage locations. When the *SUBNET card is read, the subnet name is taken from it and placed in a list which will contain the names of all subnets in the order encountered. The interface cards are read and the interface name and event number are placed in their respective tables. The *NETWORK card marks the end of interface cards and beginning of subnet activity cards. When it is read, subroutine TEST is called to read the activity cards and perform the first phase of analysis on this subnet. When the first subnet has been condensed, control returns to ASKER and the next asterisk card is read. Since other subnets follow, the procedure is repeated for each one.

When the last subnet has been condensed and control returned to subroutine ASKER, the next card read is not a *SUBNET but a *REPORT card. This indicates that all subnets have been read and condensed and causes ASKER to call on the three sorting routines, PREPAR, SSORT, and MOVE, which initiate phase two of the net-work analysis (evaluation of the control network). After this second phase has been completed, control again returns to ASKER, which interprets the *REPORT card read previously. The name of the subnet is taken from the card and located in the list of subnets. The output requests are interpreted, encoded, and placed in a second list in a position corresponding to the position of the subnet in the subnet name list. More cards are then read until a non-*REPORT card indicates that all report cards have been read. Subroutines TEST and TOPOL are then called to perform the third phase of analysis on the first subnet requested. They will also perform the first output for that subnet if it has been requested. The output request word is then checked again, and if further reports are requested for this subnet, the reporting routines are called to perform them. The process is repeated for each requested subnet in turn.

As mentioned previously, a non-*REPORT card was read by ASKER before the subnet reporting phase began. If that card was an *END BATCH card, execution terminates as soon as all report requests have been fulfilled. If it was not, another network is assumed to follow and all storages are re-initialized in preparation for its processing. In this way the analysis and reporting of the network is coordinated through the asterisk cards by subroutine ASKER.

A nonsubnetted network is handled in the same general manner (see fig. 5(b)). For such a network the *REPORT card and all other cards are read before the *NETWORK card that immediately precedes the activity cards. Upon recognizing the *NETWORK card, the network can be analyzed and reports made immediately.

Reporting

If reports other than the predecessor output report are requested for a given subnet, the output supervisory subroutine SUPER is called as soon as the subnet has been evaluated. As the final part of this evaluation, the subnet activities and associated data were placed in a file in packed form. The function of the reporting subroutines is to sort and output these activity data in various formats as specified by the PERT analyst.

For each output requested SUPER consults a table contained in the program which provides indicators of the primary and secondary keys to be used. To redefine, add, or delete output types only this table need be changed. A second table is then consulted to find the information needed to extract each key from the packed activity records. The records are then read from the file into the activity buffer, which is essentially a table of activity records in order by predecessor. The sort routines PRSCAN and SORT then determine the order of the records with respect to the keys for this particular report and output each activity in its correct order. The method by which this order is determined will be described in detail in appendix D; however, it is necessary here to mention that the activities as they are in the activity buffer are not rearranged in any way. Because of this, succeeding output formats for this subnet can be produced without having to reread the activities from the file. As each output is completed, the output supervisory routine need only obtain from the tables descriptive information for the next output format and call the sort routines again.

Of course, only a limited amount of core storage is available to be used as an activity buffer, and it is quite possible that the size of the subnet being reported may exceed the size of the buffer. The preceding discussion has assumed that this is not the case. In those cases where subnet size does exceed buffer size a merging process is required to perform the various outputs. This process is begun by reading a full buffer load of activities into the core and determining their order in the same manner as was done before. The activities are then placed in an intermediate file (in order relative to the requested format). The next buffer load is then read, ordered, and placed in a second file. If more activities remain, the third buffer load will be placed as a second set in the first file, and the fourth as a second set in the second file. Similarly, all odd-numbered buffer loads will be placed as additional sets in the first file and even-numbered loads in the second file. When all activities have been treated in this manner, the process of merging the two files begins.

The activity buffer is divided into four sections with two sections assigned to each of the files 1 and 2. Initially all sections are filled with activities from their respective files. Comparisons are then made between the activity keys from each file. At each comparison, the activity whose key is smaller is written in a third intermediate file which will be referred to as file 3. As soon as one of the sections assigned to file 1 or file 2 has been completely written on file 3, it is filled with more activities from its associated file while comparisons continue from the other section. In this way the first set of activities from file 1 and 2 are merged to form a larger set on file 3. The second sets from file 1 and 2 are then combined and placed as the first set in file 4. The third and any succeeding odd-numbered sets from files 1 and 2 will be merged and placed as additional sets in file 3; the even-numbered sets will be placed in file 4. When files 1 and 2 have been exhausted and all their activities placed in files 3 and 4, files 3 and 4 will be merged in the same way and placed in file tapes 1 and 2. The process continues until the files being merged each contain only one set of activities. At this point, rather than being written in an intermediate file, the activity whose key is found to be smaller after each comparison is written as output. The entire procedure is repeated for each output format requested. Since the activity records are in a packed form, all output is done by using subroutine OUTPUT, which must unpack and format the information associated with each activity.

As each item of information about an activity is required, a subroutine FIND is referenced which determines from a table the location and form of the item in the packed activity record. With this information the item is extracted from the record and is then correctly formatted by subroutine OUTPUT.

Updating the Master File

The master file produced by the NASA PERT Time II program is constructed in the following manner. Information for each subnet is separated by ends of file from information relating to other subnets. For each subnet, the master file contains the subnet name, a flag to distinguish between subnets and summary subnets, all interface cards, tables which give relative times calculated for every event in the subnet, control activities and their associated time durations, and the activity cards for the subnet.

The update feature makes it possible to modify any of the interface cards or activity cards and to change the subnet-summary declarations as they appear on the master file. Entire subnets may be added to or removed from the master file.

The sequence of operations in performing an update is as follows. When update cards for a given subnet are encountered on the input deck, the control routine ASKER begins reading the master file in search of the update subnet. If the subnet whose information is encountered is not the update subnet, its information is copied onto the new master tape and its control activities saved for control network analysis. This can be done since it is required that update cards for each subnet to be updated appear in the input deck in the same order as the subnets appear on the master file. It can be assumed, therefore, that any subnets passed over in searching the master file for an update subnet will not be updated themselves and that their master file information will still be valid and can be used in the second and third phases of network analysis. The copying process is continued until the update subnet is encountered; at this time control is transferred to the update subroutine UPDATE.

Subroutine UPDATE reads all interface update cards and sorts them by using subroutine USORT. The interface cards in the master file are then read and checked for order by using subroutine USORT. If the cards are unordered, they are sorted. The master file interface cards are then updated as specified by the input cards, and the resulting interfaces are placed in the new master file. When all interface updates have been performed, control passes to subroutine ACTMOD. (If no interface updates appear, the master file interface cards are copied into the new master file and control passes immediately to ACTMOD.)

Subroutine ACTMOD reads the activity update cards and calls subroutine USORT to order them. The subnet activities from the master file are compared in turn to the activity update cards. Since the update cards have been sorted and it is required that the activities in the master file be ordered by predecessor and successor event numbers, any activity having event numbers less than those on the current update card will not itself be updated. All such activities are retained in the subnet. If the event numbers of the update card are less than those of the master file activity, the update activity will be added to the subnet at a point just ahead of the master file activity. If the update card is strictly an alteration card it must refer to a nonexistent activity and will be ignored. As the third case, when the event numbers of the master file activity and update card are identical, the master file activity will be altered as specified by the update card.

Each comparison of master file activity numbers to update activity numbers results in a call to subroutine ADD. The function of subroutine ADD is to build a file containing the updated subnet. Depending on the results of the comparison, ADD will insert the original master file activity, the update activity, or an altered master file activity into the new file. After all master file activities and update cards have been processed, the new file contains the entire subnet as updated. Control then is given to subroutine TEST which reads the subnet activities from the new file and performs subnet analysis. After analyzing the subnet, TEST places its master file information in the new master file.

When all subnets have been updated or copied into the new master file, control network analysis is performed by using the control information collected while reading the old master file for updating. If reports are requested, they are then made in the usual manner.

The master file for nonsubnetted networks contains a record giving the number of activities, starts, and ends with some other network information followed by the network activities. To update such a master file, subroutine UPDATE spaces past the first record of the file and then transfers control to ACTMOD, which performs the activity updates in the same manner as described previously. After the update is complete, the network is analyzed and reports made in the usual manner for nonsubnetted networks.

Lewis Research Center,
National Aeronautics and Space Administration,
Cleveland, Ohio, March 24, 1965.

APPENDIX A

GLOSSARY OF TERMS IN LOGICAL ORDER

EVENT	identifiable instant in time
ACTIVITY	time consuming element defined by a starting (predecessor) event and an ending (successor) event
NETWORK	collection of activities and their associated events
SUBNET	subset of a network
START EVENT	event which is not a successor event
END EVENT	event which is not a predecessor event
INTERFACE EVENT	event common to more than one subnet
INTERFACE NAME	unique alphanumeric identifier for an interface event
CONTROL NETWORK EVENT	all start events, end events, and interface events relative to a particular collection of subnets
CONTROL NETWORK ACTIVITY	activity constructed by condensing to a single activity all paths between two control network events
CONTROL NETWORK	network formed by all control network activities and their associated events

APPENDIX B

PERFORMANCE DATA

The NASA PERT Time I program has been successfully run on machines of several different manufacturers and the run times have been considered favorable on all machines used thus far. The following performance data are for the first phase, PERT Time I, as recorded on an IBM 7094 running tape to tape using 729V tape drives at 800 BPI on two data channels:

Activities	Outputs	Time, min	
		PERT Time I	Mod-B
200	3	0.2	0.4
200	5	.3	.7
300	4	.4	.7
1000	5	2.5	2.7
1600	3	2.5	2.9
2830	4	7.5	6.5

Times are exclusive of load time and reflect some time savings obtained by the blocking of output at 5 lines per record.

Time studies were run by using the configuration against the NASA PERT Mod-B machine-coded program that was then still in production at Lewis. Comparative timings on the machine indicate that the program is 50 percent faster than the Mod-B PERT machine-coded program for networks under 1100 activities, requiring no output merging; equal in speed for networks between 1100 and 2100 activities, requiring a single output merge; and within 10 percent for networks over 2100 activities, requiring multiple output merges.

Since the original time study was conducted, computer configurations have been modified and the following times for a directly coupled IBM 7094 model II IBM 7040 with a disk, drum, and four model VI tape units can be reported:

Activities	Outputs	Time, min
0 to 1000	3 to 5	Under 0.5
1000 to 2000	3 to 5	Under 2.0
2000 to 3000	3 to 5	Under 5.0

APPENDIX C

TOPOLOGICAL PROCEDURES

The topological or network analyzing procedures used in the NASA PERT Time I and II programs are not the familiar topological sorting techniques used in other PERT programs. The technique used here is an application of pushdown lists or tables more commonly used in compilers and recursive routines. It is a modification of an algorithm introduced by Hans Bremer of the Goddard Space Flight Center (ref. 1).

A pushdown table is a serially ordered table of activities constructable by the addition of an activity or the removal of the most recently added activity. With the use of this table all possible paths from start events to end events can be analyzed with a minimum of bookkeeping. At the end of the analyses one wishes to be able to answer the two following questions about any given event:

(1) What is the earliest possible time when this event can be expected to occur?

(2) What is the latest possible time when this event can be allowed to occur?

The answer to the first question is called the expected time. The answer to the second question is called the allowed time. In any given path from a start event to an end event, the expected time of an intermediate event depends on earlier activities and must be forward calculated from the start event, while the allowed time depends on later activities and must be backward calculated from the end event. As subsequent paths are analyzed, expected times are altered only when they increase, since only the most pessimistic expected time is desired; allowed times are altered only when they decrease, since only the most optimistic allowed time is desired.

The general idea of the technique is as follows: For every start event determine all possible paths originating here and calculate expected and allowed times for all events on such paths. Two lists are maintained, both of the pushdown type. The first is called the pushdown list, and the second, the path list. As a path is generated, activity by activity, from a given start event to some end event, branches which are not taken are saved in the pushdown list. After expected and allowed times have been calculated for all events on the path, a new path is generated by deleting activities from the end of the former path until there is available in the pushdown list an activity that will extend the path remaining in the path list. This new path is completed in the same manner as before. When the pushdown list is exhausted, all paths originating from that given start event have been generated. When all start events have been thus treated, the network analysis is complete. (A detailed example of this appears at the end of this appendix).

The expected times are calculated as a path is completed by the use of a table of events. These event tables are used to keep expected and allowed times for each network event. The expected times are forward calculated and replace the previously calculated event expected time in the events table (TSUPE) only when the expected time now being calculated is greater. Allowed times are calculated from the last to the first event with replacement of the previously calculated allowed time in the events table (TSUPL) only when the currently calculated allowed time is smaller. When output reports are required, it becomes a simple matter to interrogate the events tables to get the predecessor and successor event times.

The following methods were used to modify the basic topological procedure and to increase its efficiency:

(1) Sequential numbering of the events eliminated events table searching. This sequential numbering also eliminated the necessity for retaining internally the actual event numbers.

(2) Retention of the activity position counter in an events table eliminates any activity table searches. This in turn eliminates the necessity for retaining internally the predecessor event of an activity.

(3) The process of calculating expected times and allowed times at all events was shortened by using the following theorems:

Theorem A: If in the course of the forward calculation of expected times for events along a specific path one computes an expected time less than the previous value, the forward calculation may be terminated without affecting final answers.

To prove this theorem, let C be the computed time and P be the previous time for event E. Then we are given $P > C$. If N is the time associated with the next activity in the path, the new computed time at its successor event is $C + N$, while the previous time is $P + N$. Clearly $P + N > C + N$, so the expected time will not be altered. In general, if S is the sum of the times for the activities between event E and an arbitrary event F on this path, $P + S > C + S$ implies that the expected time for event F will not be altered.

Theorem B: If in the course of the backward calculation of allowed times for events along a specific path, one computes an allowed time greater than the previous value, the backward calculation may be terminated without affecting final answers.

The proof for this theorem is as follows. With C and P analogously defined, we are given $P < C$. Then $P + N < C + N$ for the next backward step and $P + S < C + S$ for an arbitrary number of backward steps implies that no future allowed times will be altered were the process to continue.

A detailed example is now given. Let the activities in a table be 1-2, 2-3, 3-4, 3-6, 3-7, 4-5, and 6-5 corresponding to the network

$$\begin{tikzcd}
 & & & 4 \arrow[dr] & \\
1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] \arrow[ur] \arrow[dr] & 6 \arrow[r] & 5 \\
 & & & 7 &
\end{tikzcd}$$

with start event 1 and end events 5 and 7.

Activities 1-2 and 2-3 are successively moved from the activities table to the pushdown list and removed from there to the path list, at which time the conditions of the pushdown and path lists are as follows:

Activities	Push down	Path
1-2	(empty)	2-3
2-3		1-2
3-4		
3-6		
3-7		
4-5		
6-5		

Next, activities 3-4, 3-6, and 3-7 are added to the pushdown list. Then 3-7 is removed and added to the path list giving the following condition:

Activities	Push down	Path
1-2	3-6	3-7
2-3	3-4	2-3
3-4		1-2
3-6		
3-7		
4-5		
6-5		

Since 7 is an end event, the path in the path list is now analyzed. Activity 3-7 is removed, and activity 3-6 is moved over to the path list. Finally, activity 6-5 is moved from the activities table to the pushdown list to the path list. This completes a second path and gives the following condition:

Activities	Push down	Path
1-2 2-3 3-4 3-6 3-7 4-5 6-5	3-4	6-5 3-6 2-3 1-2

After this second path is analyzed, activities 6-5 and 3-6 are removed from the path list, and activity 3-4 is moved over to the path list. Then activity 4-5 is moved from the activities table to the pushdown list to the path list, thus completing the final path and giving the following condition:

Activities	Push down	Path
1-2 2-3 3-4 3-6 3-7 4-5 6-5	(empty)	4-5 3-4 2-3 1-2

After this path is analyzed, activities 4-5, 3-4, 2-3, and 1-2 are successively removed leaving both lists empty at the completion of the analysis.

APPENDIX D

SORTING TECHNIQUES

In preparing reports it is necessary to determine the order, with respect to several possible formats, of the activity records which make up each subnet. Because this ordering must be performed many times during the execution of any network, the procedure used must be as efficient as possible. The ordering method developed for use in NASA PERT Time I and II is now described.

The activity buffer into which the activity records have been placed constitutes a table of activities and their associated information. For each activity in the network there is an activity record and each record contains several storage words of information about its activity. Each item of activity information (predecessor and event numbers, expected and allowed dates, slack, department code, etc.) is assigned a fixed position in the activity record. With each item of information, then, can be associated two subscripts; the first refers to the position of its activity record in relation to all other records and the second to the particular item's position relative to all other activity information in the record. (An item of information pertaining to the 10th activity and which was assigned the 4th word in the activity record would have subscripts 10 and 4. The same item of information about activity 25 would have subscripts 25 and 4.)

Rather than rearranging the activity records themselves, which would be costly both in terms of execution time and core storage usage, the ordering routine rearranges their associated subscripts. At the termination of the ordering procedure there will have been produced a list of subscripts whose order indicates the order of their associated activity records with respect to the given key.

The initial phase of the process is a scanning of the activity keys to determine the extent of natural order as the records lie in core. Both ascending and descending order is detected. A list is constructed as follows: Position 1 of the list contains the number of activity records which make up the first sequence of ordered records; the sign is made negative to indicate ascending order or positive to indicate descending order. The second position refers in the same way to the second sequence and so on, so that if the activity buffer consists of n such sequences, there will be n entries in the list. If the activities lie in the buffer as shown in step 1 of table I, the list produced would be as shown in $LIST_1$. The first four activities are in ascending order as are the next three. The four activities following the second sequence, however, are in descending order so that the entry is positive. The twenty-five activity records consist of seven sequences as described in $LIST_1$.

The remainder of the ordering procedure consists of combining consecutive pairs of sequences to form half as many sequences of combined length. The smaller activity key from the first sequence is compared to the smaller from the second sequence. The subscript of the activity whose key is smaller is placed in the first position of a second list (depicted in table I step 2 as

TABLE I. - ORDERING PROCEDURE

Step 1		Step 2		Step 3	
Keys	$LIST_1$	$LIST_1$	$LIST_2$	$LIST_1$	$LIST_3$
3	-4	7	1	17	11
5	-3	10	5	8	1
7	4	6	2		12
8	-6	2	6		5
4	3		3		10
6	3		4		13
9	2		7		2
7			11		9
5			12		6
4			10		14
1			13		3
3			9		8
4			14		15
6			8		4
7			15		7
9			16		16
11			17		17
10			23		23
8			22		25
6			20		24
7			21		22
5			19		20
1			18		21
4			25		19
3			24		18

$LIST_2$). If the smaller key came from sequence 1, the key for the next activity in sequence 1 is compared to the first activity's key in sequence 2. The subscript of the smaller is placed in the second position of $LIST_2$. Comparisons continue until subscripts of all activities in one of the sequences have been placed in $LIST_2$. The subscripts from the remaining sequence are then placed in $LIST_2$ and the combining process is repeated for the next two sequences. As each pair is combined, $LIST_1$ is revised to reflect the combined length of the sequences. Step 2 shows $LIST_1$ and $LIST_2$ following the first stage of sorting whereby the seven original sequences were reduced to four. $LIST_1$ then indicates that the activities associated with the first seven subscripts form a sequence as do the activities associated with the next ten, etc. All entries in $LIST_1$ are now left positive since after the first combination pass all sequences have been constructed in ascending order.

The four sequences given by $LIST_2$ are now combined in the same manner to produce two sequences that are described by a list of subscripts in $LIST_3$. (The conclusion of this pass is represented in step 3.) Ordinarily at this point, $LIST_3$ together with $LIST_1$ would be used to produce a new list which will be placed in $LIST_2$, so that $LIST_2$ and $LIST_3$ are alternately used and overwritten. In practice, however, once the number of sequences has been reduced to two the activity whose key is smaller is simply written as output after each compare.

REFERENCE

1. Bremer, Hans: Topological Ordering Using the Pushdown Technique. Proc. NASA/Industry PERT Computer Conf., Manned Spacecraft Center, Houston (Texas), July 22-23, 1964, pp. 8-1 - 8-24.